

On the consensus algorithm of IPchain

Bachelor Thesis

Software Engineering Specialization

Author: Jan Manrique Roca

Director: Albert Cabellos Aparicio

Codirector: Jordi Paillisé Vilanova

Abstract

A consensus algorithm is the core of a blockchain, a technology that allows the development of flexible trust relationships in distributed systems.

This project aims to design and develop a consensus algorithm for IPchain, a blockchain that will act as a security layer for the inter-domain routing protocol BGP.

To do it, we will study the most relevant consensus algorithm at the time, and use the acquired knowledge to build a customized consensus algorithm that fits the needs of IPchain.

Index

Abstract	2
Index	3
1. Context and scope of the project	6
1.1 Context	6
1.1.1 Introduction	6
1.1.2 Stakeholders	7
1.2 State-of-the-art	8
1.2.1 RPKI	8
1.2.2 Papers on the topic	8
1.2.3 Route Monitoring Services	9
1.2.4 Internet Routing Registry	9
1.3 Scope of the project	10
1.3.1 Objectives	10
1.3.2 Possible obstacles	10
1.4 Methodology	11
1.4.1 Development methodology	11
1.4.2 Used tools	11
2. Project planning	12
2.1 Task description	12
2.1.1 Project management	12
2.1.2 Study of Proof of Work algorithms	12
2.1.3 Study of Proof of Stake algorithms	13
2.1.4 Research of lesser-known consensus algorithms	13
2.1.5 Decision of best consensus algorithm	13
2.1.6 Implementation of consensus algorithm	13
2.1.7 Test of consensus algorithm	13
2.1.8 Elaboration of the report and defense preparation	14
2.2 Time estimation and task sequencing	14
2.2.1 Time estimation per task	14
2.2.2 Task sequencing and dependencies	14
2.2.3 Gantt chart	15
2.3 Alternatives and action plan	15
2.3.1 Implementation of consensus algorithm is harder than expected	15
2.3.2 Consensus algorithm doesn't behave as expected	16
2.3.3 Discovery of better consensus algorithm after implementation has started	16
2.3.4 Not enough server computers available to replicate real case scenario	16
2.4 Project completion	16

3. Budget and sustainability	17
3.1 Self-assessment	17
3.2 Cost identification	18
3.2.1 Human resources	18
3.2.2 Indirect costs	18
3.2.3 Direct costs	19
3.2.4 Contingency	21
3.2.5 Incidents	21
3.2.6 Budget	22
3.3 Sustainability	22
3.3.1 Environmental sustainability	22
3.3.2 Economic sustainability	23
3.3.3 Social sustainability	23
3.3.4 Sustainability matrix	24
4. Technical background	25
5. Researched consensus algorithms	27
5.1 Proof-of-work	27
5.1.1 Explanation	27
5.1.2 Pros	28
5.1.3 Cons	29
5.2 Proof-of-stake	29
5.2.1 Explanation	29
5.2.2 Pros	30
5.2.3 Cons	30
5.3 Federated Byzantine Agreement	31
5.3.1 Explanation	31
5.3.2 Pros	34
5.3.2 Cons	34
6. Design	34
6.1 Discussion of researched algorithms	34
6.2 The protocol	36
7. Implementation	37
7.1 Threshold signature	38
7.1.1 Explanation	38
7.1.2 Implementation	39
7.2 Distributed Key Generation	41
7.2.1 Explanation	41
7.2.2 Implementation	42
7.3 Pseudorandom number generation	46

8. Results	48
8.1 Performance	52
8.1.1 Time to setup DKG	53
8.1.2 Time to generate random number	54
8.2 Parameter recommendations	54
8.2.1 Duration of the DKG setup	54
8.2.2 Amount of participant members on DKG	54
8.2.3 Threshold	54
9. Conclusions	55
10. Future improvements	56
References	57

1. Context and scope of the project

1.1 Context

1.1.1 Introduction

This project is a Bachelor Thesis developed for the Bachelor Degree in Informatics Engineering taken at Facultat d'Informàtica de Barcelona (FIB) of Universitat Politècnica de Catalunya (UPC).

The internet is a network formed by a lot of different networks, which are called domains.

A domain is a group of computers that are managed by the same organization (e.g: google, amazon, ...). If we try to visit a website hosted in a computer in a different domain, we won't be able to visit it unless we have Internet connection. This happens because our Internet Service Provider (ISP) provides us access to other domains through inter-domain routing. Inter-domain routing means that a computer can tell which domains it has to go through in order to reach a specific domain.

The inter-domain routing protocol used nowadays is known as BGP (Border Gateway Protocol). BGP divides the Internet into Autonomous System (AS), networks managed by network operators that contain one or more domains. Thanks to BGP, an AS can announce to other AS's the domains it contains, but also receive this information from other AS's. With this information, each AS can create a table indicating the AS where each domain is found.

The BGP has been working on the Internet since 1994, but some security issues have been raised since. BGP is not a secure protocol, and an accidental misconfiguration by the network operator or the modification of AS announcements by a malicious attacker controlling an AS can wreak havoc on the Internet. We can find several examples of this kind of attack [1-3].

IPchain is a solution to BGP security flaws. It's a blockchain that stores allocations and delegations of IP addresses with the aim of easing the deployment of secure inter-domain routing systems, which prevent traffic redirections [4].

Blockchain is a technology that leverages cryptography and distributed ledger technology (DLT) to provide its users of flexible trust models, overriding the necessity of a central authority.

A consensus algorithm is the core of a blockchain, specifying how all the correct nodes in the chain (nodes that aren't malfunctioning or have malicious intentions) will achieve the same

state. In other words, it's the algorithm that will allow the blockchain to make decisions without depending on a central authority.

This project aims to study different consensus algorithms, reasoning which is the most appropriate for IPchain. Once a consensus algorithm has been chosen, it will be implemented on the current IPchain codebase and tested to see how it performs.

1.1.2 Stakeholders

Because IPchain improves the current inter-domain routing protocol of the Internet, it affects several people:

- **Network operators:** BGP configuration is hard, tedious and error-prone. It's also a juicy target for a malicious attacker. A single misconfiguration could lead to huge problems. IPchain can drastically reduce the impact of both, providing a safety network for network operators.
- **Internet users:** Users will be benefited of the implementation of secure inter-domain routing, as this would result in a safer Internet.
- **Cisco Systems:** Cisco Systems will be benefited because it will be able to produce and sell routers which have IPchain capabilities, increasing its market and competitiveness.
- **Computer Science Community:** Blockchain has been severely criticized because it hasn't been able to live to the expectations. Its adoption rate is slow, and we have yet to see a blockchain-based solution that provides real-value. We expect IPchain to become an example of success on the usage of blockchain.

1.2 State-of-the-art

1.2.1 RPKI

It should be noted that the Internet Engineering Task Force (IETF) is already aware of these problems and has designed a solution to inter-domain routing security. This solution is the Resource Public Key Infrastructure (RPKI) [5], which relies on a central authority that records the legitimate owners of IP prefixes and AS numbers (the identifier of an AS).

But something must be wrong because adoption rate of RPKI is quite slow. On Sept 2018, ~88000 AS were identified[6], but only ~9000 are protected by the RPKI [7], a 10% of all the AS. The reason of this has been analyzed by the literature [8-9] but can be summarized in:

- Central authorities hold control of RPKI resources, but IP addresses are a critical asset of RPKI users, and they would like a higher degree of control while maintaining security provided by RPKI.
- PKI are hard to manage, require trained staff and financial investment.
- RPKI expose business relationships through peering agreements.

1.2.2 Papers on the topic

Similar projects have been discussed on the literature. We will discuss them in this section.

The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet [10] argues that the internet needs a distributed and tamper-resistant resource management framework which cannot be subverted by any single entity, and the best possible implementation is blockchain. Thus, they propose to use blockchain to secure BGP and DNS.

BGPcoin: A Trustworthy Blockchain-based Resource Management Solution for BGP Security [11] plans to do the same thing that IPchain does, but while the paper was released on 2017, nothing has been heard off BGPcoin since. Their implementation was going to be based on Ethereum, which provides less flexibility as the consensus algorithm can't be modified to adapt to the needs of BGP security.

An experiment in distributed Internet address management using blockchains [12] presents InBlock, an experiment to use blockchain for the validation of domains in BGP. It's supposed to be a starting point to explore the technology and inform future directions in this area. It's also based on Ethereum.

Blockchain-based Public Key Infrastructure for Inter-Domain Secure Routing [13] presents SBTM (Secure Blockchain), a general and flexible management framework for inter-domain routing based on a blockchain protocol. This project has the same objective as IPchain but is completely based on proof of work (which is known to have several problems). Its aim is to actually test the use of blockchain on inter-domain routing security instead of looking for the best consensus algorithm.

1.2.3 Route Monitoring Services

Some companies such as BGPMon [14] and ThousandEyes [15] sell subscription-based solutions that provide real-time feedback. These companies maintain one or several nodes distributed over the Internet that receive and review BGP information. When a user is subscribed, the company will track the user's domain information on their network nodes and if an incoherence is detected, they will warn the user, giving details about the problem.

While this is a functional solution, it has a few problems:

- Instead of preventing problems, it warns about them once they have happened. Even if the problem is fixed fast, it could take a while to propagate through the Internet, potentially resulting in losses for the affected parts.
- User security depends directly on a third party, which can be compromised, change the way their information is gathered, etc. It's preferable to be able to handle security by yourself.

1.2.4 Internet Routing Registry

The Internet Routing Registry [16] is a globally distributed database that contains routing information. Its purpose is to ensure the stability and consistency of Internet routing by sharing information between network operators. Network operators, who are allowed modification of the IRR (Internet Routing Registry), must update the information of their own networks manually so it reflects the status of their AS routing configuration. IRR is mostly used to validate information provided by other AS.

Some problems with this solution are:

- It's tedious because information update and validation must be performed manually.
- It's not secure, because a network operator (or someone with its credentials) can upload invalid information (either accidentally or with malicious intentions).
- It doesn't prevent routing problems, but instead helps to debug them once they have already happened.

Sometimes network operators use phone calls between them in order to complement validation through IRR.

1.3 Scope of the project

1.3.1 Objectives

As it's been noted earlier, the objective of this project is the development of the most appropriate consensus algorithm for IPchain, a blockchain solution for inter-domain routing security. To do so, the following steps will be followed:

- Study the currently known, tested and proved consensus algorithms that are used in blockchain solutions, either in research or by companies. Identify the trade-offs, advantages and disadvantages of every consensus algorithm. Once all representative consensus algorithm have been studied, determine which one fulfills the requirements of IPchain the most.
- Once a consensus algorithm has been chosen, study the existing IPchain codebase and implement it on IPchain. We will implement an extensible module that manages consensus of the blockchain node following the principles of software engineering. It must be possible to easily add a new consensus algorithm, in case a new and more interesting solution appears in the future.
- Test the implemented consensus algorithm, verifying it works correctly on real emulation of the BGP protocol and the non-functional requirements are fulfilled (block-time, performance, etc...). We must also ensure that there are no bugs or unexpected behaviors.

1.3.2 Possible obstacles

Some obstacles can appear during the development of the project, affecting its temporal planning:

- Chosen consensus algorithm is hard or impossible to implement with our resources. If we find this obstacle, we will evaluate time left and complexity of the algorithm and decide if we implement a simplified but functional version of the algorithm or we provide a theoretical explanation of the algorithm on the project report.
- We don't have enough resources to emulate a real case scenario of the BGP protocol. In this case we will have to generate a representative fragment of the AS networking and apply the BGP protocol in this subnet. The results taken will have to be extrapolated to the real case scenario.
- Problems with used computer and loss of information. In order to avoid this problem we will use online services and store the code in a Version Control System (VCS).

1.4 Methodology

1.4.1 Development methodology

Given the fact that we are working on an unknown project and we don't know what problems we'll have, we've decided to use an agile methodology. That's why we've decided to use Kanban. We will have a backlog with all the tasks that need to be completed and we will have weekly meetings to check and discuss the development of the project, making changes if required.

We will keep track of the current status of the project and compare it to the project planning explained in section 2, reasoning how far are we from completing the objectives of the project and, if needed, readjusting the time dedication of every task.

For example, if we have exhausted the assigned time of consensus algorithms we will look at the solutions that have been studied and decide if we use one of those or dedicate some more time to researching consensus algorithms, which will have to be sacrificed from a different task (such as testing).

1.4.2 Used tools

We will use Trello to manage the backlog and the time dedication of all the tasks, checking on the weekly meeting how far are we from our objectives and making task changes if required.

We will also use Github as a VCS where we will store and manage the code.

2. Project planning

2.1 Task description

2.1.1 Project management

When developing a Bachelor's Thesis, the faculty requires its students to course a project management subject called Gestió de Projectes (GEP). The focus of this subject is to guide the students on the planning of the project, helping them specify what the problem is and how will it be solved.

The deliverables of this subject and its time estimations are:

- Context and scope of the project (12 hours)
- Project planning (8 hours)
- Budget and sustainability (8 hours)
- Review of bachelor's thesis competences (10 hours)
- Oral presentation and final document (20 hours)

The expected dedication of the student for GEP is 75 hours, while the deliverables have been estimated to take 58 hours. There are 17 hours remaining that will be used to study the material required for every deliverable.

In order to complete this task, we will require: a computer, Google Drive (to store and edit the deliverables online), Libre Office (to complete the rubrics associated with every deliverable), Firefox (to visualize websites and PDFs) and Atenea (to upload the deliverables).

2.1.2 Study of Proof of Work algorithms

We will study algorithms that rely on proof of work to achieve consensus on blockchains. This kind of consensus is very interesting because it's the first that achieved decentralized consensus. It's mainly known for its application in Bitcoin, but has some drawbacks that might make it unappealing for our use case. We will have to study these drawbacks and look for modifications of classic proof of work that solve those drawbacks that could matter to us. This task will require 20 hours of work, and a computer with access to the Internet will be required to complete it.

2.1.3 Study of Proof of Stake algorithms

We will study algorithms that rely on proof of stake to achieve consensus on blockchains. This is the kind of consensus that is expected to fix the problems of proof of work. They are based on a different premise, and because of this, they have a different set of drawbacks. We have to keep in mind this kind of algorithms haven't seen a lot of real world usage yet, and because of this, there are a lot of different implementations with different trade-offs. This task will require 30 hours of work, and a computer with access to the Internet will be required to complete it.

2.1.4 Research of lesser-known consensus algorithms

Although proof of work and proof of stake are the most known consensus algorithms, they are thought for blockchains that handle cryptocurrencies. Blockchain is a solution that is being tested in different areas and other solutions might exist to the consensus problem. In this task, we will explore those solutions. This task will require 30 hours of work, and a computer with access to the Internet will be required to complete it.

2.1.5 Decision of best consensus algorithm

Once we consider we have researched enough on consensus algorithms, we will have to compare all the solutions and decide which one are we going to use. We won't only consider the one that fits the best, but we will also have to keep in mind robustness (how reliable the algorithm is and how much has it been tested), complexity of the solution and flexibility (how well can it scale). This task will require 20 hours of work and a computer.

2.1.6 Implementation of consensus algorithm

After deciding the consensus algorithm we want to use, we will have to implement it on IPchain. Because a code base already exists, we will have to study the code first and decide how to modify it following the principles of Software Engineering. This task will require 150 hours of work, and we will require: Docker to create containers that act as blockchain nodes, a computer that will host the blockchain nodes, a computer where the project will be developed and a Git server (Github) where the code will be hosted.

2.1.7 Test of consensus algorithm

Once the consensus algorithm has been implemented, we will have to verify that it fulfills the IPchain requirements. We will need to initialize a blockchain and add enough nodes to replicate a real case scenario. Then, we will have to gather several statistics (such as block insertion time, tolerance to malicious nodes, ...) and reason if they are good enough. This task will require 150 hours of work and will require: Docker to deploy containers that act as blockchain nodes, four server computers that will host the blockchain nodes and a computer to monitor the blockchain and gather statistics.

2.1.8 Elaboration of the report and defense preparation

Once the project is completed and tested, all the work that has been done must be documented in a project report, which will be used to defend the project in front of an evaluation committee composed of professors of FIB. This task will require 50 hours of work and will require a computer with access to the Internet, because Google Drive will be used to elaborate and host the document online.

2.2 Time estimation and task sequencing

2.2.1 Time estimation per task

<i>Task</i>	<i>Estimated hours</i>
Project management	75
Study of Proof of Work algorithms	20
Study of Proof of Stake algorithms	30
Research of lesser-known consensus algorithms	30
Decision of best consensus algorithm	20
Implementation of consensus algorithm	150
Test of consensus algorithm	150
Elaboration of the report and defense preparation	50
Total	525

2.2.2 Task sequencing and dependencies

<i>Task</i>	<i>Predecessor task</i>
Project management	-
Study of Proof of Work algorithms	-
Study of Proof of Stake algorithms	-
Research of lesser-known consensus algorithms	-

Decision of best consensus algorithm	<ul style="list-style-type: none"> - Study of Proof of Work algorithms - Study of Proof of Stake algorithms - Research of lesser-known consensus algorithms
Implementation of consensus algorithm	Decision of best consensus algorithm
Test of consensus algorithm	Implementation of consensus algorithm
Elaboration of the report and defense preparation	Test of consensus algorithm

2.2.3 Gantt chart

The following Gantt chart shows the project planning and the expected completion time, as well as the task dependencies.



2.3 Alternatives and action plan

2.3.1 Implementation of consensus algorithm is harder than expected

Because consensus algorithms are a trending topic, there is a lot of research papers in the area, but usually research papers explain the theory but don't append the implementation. It is possible that we decide to implement a consensus algorithm we discovered through a research paper and find some unexpected complexity we don't know how to approach. In this case, we would have to contact the authors of the paper to discuss their solution and ask for some guidance, which could delay the project a bit. This complication would cause a one/two weeks delay (30 to 50 hours), depending on the time it takes us to reach the authors of the paper.

2.3.2 Consensus algorithm doesn't behave as expected

Because some consensus algorithms can be very experimental, it's possible that once we start testing the implemented consensus algorithm we discover it doesn't behave as expected. In that case, we wouldn't have time to implement a different algorithm, so we would try to fix the one that is already implemented. If we realize we won't be able to fix it in time, we will continue the project, documenting what went wrong, why, if it's possible to fix it and how would we do it. This complication would cause a delay of one/two weeks.

2.3.3 Discovery of better consensus algorithm after implementation has started

Given the amount of information about consensus algorithms, it's easy to overlook or miss information. It's also possible that a new consensus algorithm is revealed while we are developing this project.

The discovery of new consensus algorithms won't affect us unless we have started the implementation. In that case, we would evaluate time left and how close are we from completing the implementation. If we think we have enough time, we will switch to the new consensus algorithm, otherwise we will simply comment it on the project report. This complication will only be approved if it doesn't cause a delay bigger than two weeks.

2.3.4 Not enough server computers available to replicate real case scenario

While the amount of Autonomous System on the Internet isn't specially big, we don't really know how computationally intensive will the consensus algorithm be. It's possible that we don't have enough computers to emulate a real case scenario. In that case, we will spin up as many nodes as possible and gather our statistics. Then, we will extrapolate them to the amount of nodes we were expecting to use. This complication would cause a 5 days delay.

2.4 Project completion

The project must be presented at the end of January. As we can see on the Gantt chart at section 2.2.3, we have enough time to complete the project safely. We also have enough spare time which would be used in case we had any unexpected problem.

3. Budget and sustainability

3.1 Self-assessment

Before studying the assigned material for GEP, I wouldn't pay a lot of attention to social and economical sustainability of the projects I worked on. I can't say the same about environmental sustainability because it's a very trending topic lately on technology projects, and it's something I had already thought about on this project. Now I feel like I know what I have to think about when taking care of the sustainability of a project, and I can make a decent analysis.

Regarding environmental sustainability, I think I can easily identify key characteristics that define the environmental impact of a project and provide ideas and solutions that make the project more sustainable. I have also learned how to measure the sustainability of a project.

Regarding social sustainability, I can now take into account the impact that a project can have in society. I find it easier to think about how it benefits people, and I think I need to work on reasoning if it can be detrimental for any affected part, and possible misuses of the project. I also have difficulties on deciding if a project will improve the well-being of society when taking pros and cons in consideration.

Regarding economical sustainability, I've learned how to measure the cost of implementing a project, while taking both human and material resources in consideration. I have also gained perspective on how uncertainty affects the budget of a project. I need to think more on improving the viability of projects during their useful lives.

Overall, I think I have achieved the objectives of GEP and if I keep developing these new acquired abilities I will be able to develop truly sustainable projects.

3.2 Cost identification

3.2.1 Human resources

Role	Hours of work	Salary/Hour	Estimated cost
Project author	525	10	5250 €
Project director	100	16	1600 €
Project codirector	100	16	1600 €
<i>Total Cost</i>			<i>8450 €</i>

The hours of work of the project author were calculated previously in this document. The hours of work of both directors have been calculated assuming 8 hours of work per week: they will devote 4 hours to the weekly meeting and 4 hours to personal research.

3.2.2 Indirect costs

Product	Units	Price/Unit	Percentage of use	Estimated cost
Internet	4 months	40 €/month	40%	64 €
Electricity	4 months	50 €/month	30%	60 €
Local	4 months	300 €/month	100%	300 €
Transport	20 days	20 €/day	100%	400 €
<i>Total</i>				<i>1724 €</i>

- Transport: The author of the project has to take a RENFE train to commute from Reus to Barcelona for the weekly meeting. Metro fees have also been added.
- Local: It will be supplied by the Departament d'Arquitectura de Computadors of FIB.
- Electricity and Internet: Based on the fees paid by the project author, applying an approximate percentage of resource dedication.

3.2.3 Direct costs

In this section, we will detail the resources required to develop every task defined in the previously in this document.

Task	Units	Price/Unit	Useful life	Amortization	Price
<i>Project Management</i>					
Laptop	1	600	4 years	$(600/7680) * 75$	5.86
Google Drive	1	-	-	0	0
Libre Office	1	-	-	0	0
Mozilla Firefox	1	-	-	0	0
<i>Study of Proof of Work algorithms</i>					
Laptop	1	600	4 years	$(600/7680) * 20$	1.56
<i>Study of Proof of Stake algorithms</i>					
Laptop	1	600	4 years	$(600/7680) * 30$	2.34
<i>Research of lesser-known consensus algorithms</i>					
Laptop	1	600	4 years	$(600/7680) * 30$	2.34
<i>Decision of best consensus algorithm</i>					
Laptop	1	600	4 years	$(600/7680) * 20$	1.56
<i>Implementation of consensus algorithm</i>					
Laptop	1	600	4 years	$(600/7680) * 150$	11.72
Server computer	1	-	-	0	0
Github	1	-	-	0	0
Docker	1	-	-	0	0
<i>Test of consensus algorithm</i>					

Task	Units	Price/Unit	Useful life	Amortization	Price
Laptop	1	600	4 years	$(600/7680) * 150$	11.72
Server computer	1	-	-	0	0
AWS EC2 General Purpose Large	60	0.17	-	-	10.2
Docker	1	-	-	0	0
<i>Elaboration of the report and defense preparation</i>					
Laptop	1	600	4 years	$(600/7680) * 50$	3.91
Google Drive	1	-	-	0	0
<i>Total</i>					<i>51.21</i>

Amortization is based on working hours per year, with 40 work hours per week and 4 vacation weeks per year. The formula used is: $40 \text{ (hours per week)} \times (52 \text{ (weeks in a year)} - 4 \text{ (vacation weeks)})$.

A server computer will be provided by the Departament d'Arquitectura de Computadors of FIB. It has already been amortized, so it doesn't have any cost.

Three more server computers will be required, which will be acquired through Amazon Web Services. We will need each server for 20 hours, so we will need 60 hours of Amazon Web Services EC2 time. AWS is paid per hour.

3.2.4 Contingency

Given the fact that blockchain is a hot topic and a lot of our project is based on research papers, there is a lot of uncertainty regarding the project. Because of this, we have assumed a contingency of 15%.

Concept	Cost	Contingency
Human resources	8450 €	1267.5 €
Indirect costs	1724 €	258.6 €
Direct costs	51.21 €	7.68 €
<i>Total</i>	<i>10225.21 €</i>	<i>1533.63 €</i>

3.2.5 Incidents

- Two weeks delay: In the project planning we mentioned the possibility of certain problems and its impact on the project. The average delay time was of one/two weeks. We calculate the cost of this incident using the salary described in section 3.2.1. The probability of this incident is 25%.
- Computer fault: If the laptop is broken during the development of the project, it will have to be replaced by a new one. The probability of this incident is 5%.

Incident	Probability	Units	Price	Cost
Two weeks delay	25%	54 hours /author 8 hours/director	10 €/h 16 €/h	175 €
Computer fault	5%	1	600 €	30 €
<i>Total</i>				<i>205 €</i>

3.2.6 Budget

This is the final budget, adding all the costs previously described.

Concept	Cost
Human resources	8450 €
Indirect costs	1724 €
Direct costs	51.21 €
Contingency	1533.63 €
Incidents	205 €
<i>Total</i>	<i>11962,84 €</i>

3.3 Sustainability

It will be studied answering some questions proposed by the GEP teachers.

3.3.1 Environmental sustainability

- Have you estimated the environmental impact of the project? Did you plan to minimize its impact, for example, by reusing resources?

We will use a laptop (0.07 kWh) for 525h and servers (0.15 kWh) for a total of 80h. That's a total of 48.75 kWh. No measures have been taken to reduce the impact, as the amount of resources used is very small already.

- How is currently solved the problem that you want to address (state of the art)?

The problem is addressed in two different ways. The first way is through the maintenance of a big database known as IRR that contains a lot of information regarding inter-domain routing. The second way is through companies that provide monitoring services, sending alerts to network operators when a suspicious behavior is detected.

- How will your solution improve the environment with respect other existing solutions?

It will reduce the amount of operative computers, as neither the monitor servers nor the IRR database will be needed, while the amount of operational computers will be the same. These operational computers will slightly increase their computational load due to the consensus algorithm and the route verification. Overall, the amount of consumed energy will be reduced.

3.3.2 Economic sustainability

- Have you estimated the cost of developing the project (human and material resources)?

Yes, the estimation can be found on section 3.2. In order to reduce the cost, we have chosen cheap AWS servers and reduced the amount of hours that directors will work on the project. All in all, we believe this is a cheap project given the benefits it provides.

- How are currently solved economic issues (costs...) related to the problem that you want to address (state of the art)?

Monitor services provided by specialized companies are subscription-based, so the cost of the servers is paid with the benefits obtained from client subscription. The IRR is maintained by the Regional Internet registry (RIR) and its costs are paid with the income obtained through the membership fee of all the Local Internet Registries (LIR).

- How will your solution improve economic issues (costs ...) with respect other existing solutions? Since the amount of operational computers and computational load will be reduced, less energy will be wasted and the costs of the solution will be reduced. Because the system is decentralized and there is no server, the cost of using IPchain will be paid directly by their users in the form of consumed energy.

3.3.3 Social sustainability

- How do you think that will the development of this project contribute to you?

The development of this project will be my first contribution to a research paper. The experience of contributing to it could change my career (orienting it to research contributions).

- How is currently solved the problem that you want to address (state of the art)?

The problem is addressed in two different ways, as explained earlier. The IRR solution requires manual updating of the BGP information by the network operator. The monitoring services require integration and dependency of a third-party.

- How will your solution improve the quality of life (social dimension) with respect other existing solutions?

It will improve the quality of life of network operators, reducing their workload. They won't have to update their BGP information manually on IRR nor depend on a third-party. Also, the impact of a misconfiguration or a successful malicious attack will be severely reduced, thus the urgency of a fix will be decreased and so will the network operator required availability.

- Is there a real need for the project?

Yes. Network operators are disposed to interact with third party and manually update the IRR in order to ease the debugging of inter-domain routing problems. This project will make their job easier and severely reduce the impact of any mistake. Currently inter-domain routing problems are more frequent than desired and any misbehaviour of the BGP protocol will result in a lot of trouble and extra work hours for network operators.

3.3.4 Sustainability matrix

The sustainability matrix obtained from the results of the previous sections is:

Environmental	Economic	Social
8/10	8/10	9/10

4. Technical background

Before diving into the technical part of the project, we think it is interesting to give our definition of blockchain and consensus algorithm.

The amount of projects claiming to use blockchain technologies is rising, but the word itself is becoming a buzzword due to the amount of misinformation surrounding it. No one seems to agree on which qualities are essential in order to call something a blockchain.

Bitcoin is a widely-known cryptocurrency that is considered the first blockchain, and we believe it's implementation perfectly adjusts to the term, so we are gonna use it to explain what blockchain actually means.

Since Bitcoin is expected to be used as a cryptocurrency, the system will have to keep track of the amount of currency owned by every user, as well as update it when a transaction between users occurs, just like a bank would do. This record is what we understand as state of the system.

The first thing we need to know, is that in Bitcoin there is no central machine maintaining the state of the system; instead, a distributed ledger is used. What this means is that the system contains an undetermined amount of nodes, and each one will have a private copy of the state that we'll be maintained by itself. The nodes will be interconnected using a peer-to-peer (P2P) network, and a **consensus algorithm** will be used so the nodes can agree on how to update their state. This kind of system is known as Replicated State Machine.

When users send transactions to the system, these are broadcasted through P2P. Every node will receive the transaction, verify that it's valid (validate sender with public-key cryptography, check the balance of the sender, ...) and store it in a list of transactions pending to be applied. The list of transactions can be different for every node, so we need a way to know which transactions are being handled when nodes try to reach consensus, otherwise nodes could apply the same transaction twice. This is where blocks come into play.

A block is a package that is created by a node and contains interesting information. For this explanation we will focus on two fields: a list of transactions and a hash of a block. The transactions from the list are validated and deemed correct by the node, and the hash of a block is the hash of the last block considered valid by the node. This hash links all valid blocks all the way back until the first one, resembling a chain, and that's where the term **blockchain** comes from (Figure 1). There is only one special block, the first block. Since it can't point to any other block, it is usually hardcoded and it's called genesis block. Every other block will have the same structure.

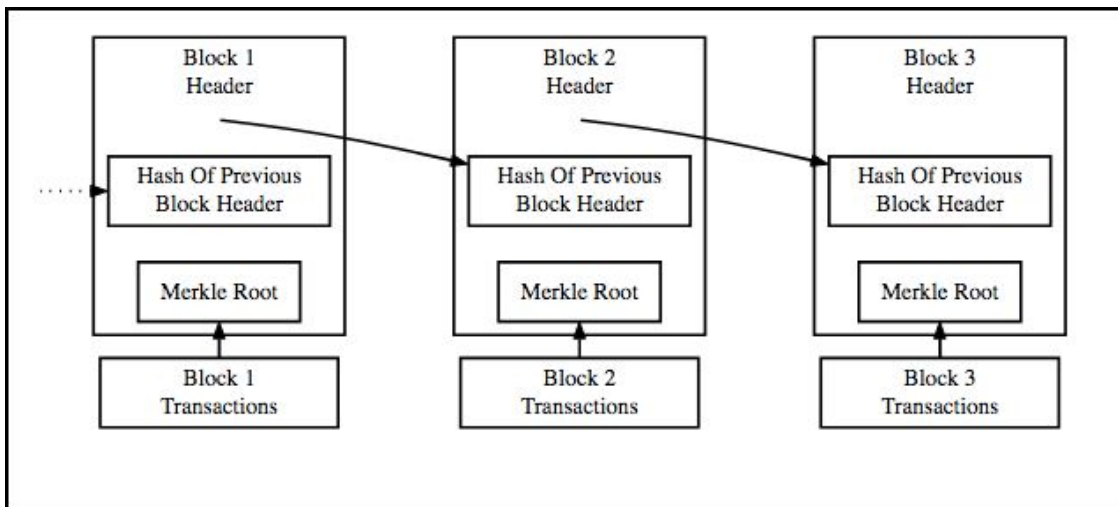


Figure 1: Simplified Bitcoin Block Chain

Who creates a block, when does he do it and how do other nodes accept a block is defined by the consensus algorithm, so we won't explain it here. All we need to know right now is that eventually, all nodes will receive a certain block which by the protocol standards is correct, and they can either accept it or refuse it.

If a node accepts a block, it will remove the transactions included in the transaction list of the block and apply them to its current state. If he doesn't accept the block, he won't be able to propose to or accept blocks from nodes that have accepted it, as the hash contained in that block will be seen as invalid.

When a new node connects to the network, it needs to get the current state, but as we have seen, it is possible to have different states for the same blockchain (it can have forks), so the implementation must provide a way to decide the best of two chains. This is usually also related to the consensus algorithm. In Bitcoin's case, the longest chain is picked.

Using blocks makes the system transparent, since the chain contains all the transactions that have happened since the launch of the system. Anyone can revise any transaction and validate the current state of the system.

It also makes the system tamper-resistant, because modifying a transaction in a block implies modifying all the blocks from the one that includes the transaction to the most recent one. Even though block creation is defined by the consensus algorithm, it usually takes a big amount of time. Because of this, in Bitcoin transactions aren't confirmed until the block they belong to is a certain number of blocks deep. Blockchains are not immutable, but tampering with them requires a huge effort.

As a conclusion, we understand a blockchain as a system where transactions are gathered in blocks that are secured into the chain using cryptography, and it is designed to be tamper-resistant and produce immutable records. That said, there are some examples of distributed ledgers where transactions are not organized in blocks.

The most interesting part of blockchains are the way they can allow distributed systems to reach consensus without the presence of a trusted party. In a way, they are a tool that allows systems to express customized and flexible trust relationships. We understand a consensus algorithm as the protocol that lies at the core of a blockchain and defines the policies of creation and insertion of blocks in the blockchain, driving forward the system.

5. Researched consensus algorithms

In this section we will explain the consensus algorithms that we have studied. There are a lot of algorithms out there, and they are usually complex pieces of technology that require a certain amount of time to fully understand them and the problem they solve.

Because of this, we were not able to study as many algorithms as we would have liked to, and limited ourselves to representative examples of every type of consensus.

Since IPchain aims to be a public blockchain, we haven't studied consensus algorithms used in permissioned or private blockchains (where blocks can only be created by a select group of nodes).

5.1 Proof-of-work

Proof-of-work (PoW) is the most popular algorithm because it is currently used by both Bitcoin and Ethereum, the most popular cryptocurrencies and blockchain implementations. It was first described by the Bitcoin whitepaper [17]. While they both implement PoW in a different way, the basis of their consensus is the same.

5.1.1 Explanation

In PoW, the consensus is driven by complex computational problems. In this model, a node can create a block whenever he wants, but there's one condition that the block must fulfill in order to be accepted by everyone else: the hash of the block must be under a certain number, the target. This model is used because generating the correct hash can be incredibly hard, but it's also very easy to validate.

Since the output of a hash is unpredictable, the only thing the node can do to create a block with a hash under the target is randomly create blocks until he gets the desired hash. In order to simplify this process, blocks incorporate a 4-byte file called nonce. All nodes have to do once they have created a block is increment the nonce of the block until the hash is under the target. Because there's a chance no nonce value provides the desired hash, nodes can also alter the timestamp field to find a valid combination. The process of finding the correct hash is called mining.

The target is a variable number, and is used to enforce a block time. The block time is the average time that it takes to insert a new block in the blockchain. The target will be directly influenced by the insert time of the last block: if it took longer than expected the target will be increased, while if the insertion time took too short, the target will be reduced.

Block time is a variable defined by the developers of the system. Using shorter block times means transactions will be processed faster, but the blockchain will be formed by more blocks, which will make the blockchain heavier to process. Using higher block times decreases the probability of two nodes creating a valid block at the same time and makes creation of fake blocks even harder.

Since insertion of blocks is so costly in PoW blockchains, when comparing two chains in order to decide which one is valid, nodes will always pick the longest one, because it is the one that has been harder to compute. There's an underlying assumption here: as long as there are more well-behaved than ill-behaved nodes, the longest chain will always be correct.

The result of the PoW algorithm is that approximately every "block time", one node will be elected to create and propose a block.

5.1.2 Pros

- 1) PoW systems are the most tested ones in the blockchain ecosystem. Bitcoin has been around since 2008 and the results have been satisfactory.
- 2) The cost of solving complex computational problems to create blocks deters users from behaving incorrectly, since they might waste a lot of resources without obtaining any result. Blocks will still be created even if there are no transactions pending to be processed, so rewriting the chain is very costly. A malicious party would have to create blocks faster than the entire system in order to rewrite the transaction history, and they would still require a lot of time if they want to rewrite a deep enough block.

5.1.3 Cons

- 1) The main drawback of PoW is the amount of energy it consumes. Given the public and distributed nature of Bitcoin, it is very hard to get conclusive data, but some studies have pointed that it may consume as much energy as Ireland [18].
- 2) Because of the assumption that 51% of the nodes are well-behaved, it would be possible by a resourceful party to acquire enough computational power to affect the regular functioning of the system. This is called a 51% attack.
- 3) Block times in PoW are not constant. This happens because block time is decided randomly. Although improbable, a node could generate a valid hash in seconds or days. This can be annoying for the users of the blockchain because transactions could take a long time to reflect in the state of the system.
- 4) It isn't as decentralized as initially thought, since actors can invest in better equipment in order to build more blocks than others.

5.2 Proof-of-stake

Proof-of-stake (PoS) was first proposed in a Bitcoin forum [19], and has been adopted since by some cryptocurrencies such as Peercoin, NXT, ...

Ethereum, the most popular blockchain after Bitcoin, is planning to migrate to PoS eventually. It is seen as the most mainstream solution to the problems of PoW.

5.2.1 Explanation

PoS tries to replicate the behaviour of PoW while removing the inefficiency of mining. If the summarized behaviour of PoW was "pick a leader every block time", PoS is gonna try to do the same without using complex computational problems.

If in PoW nodes had more changes to be elected according to their computing power, in PoS they will be picked according to something called "stake", that represents how many resources they have invested in the blockchain. In cryptocurrencies, the stake is the proportion of coins held by that node. In IPchain, the stake could be, for example, the amount of IP prefixes a node holds. The theory behind it is that the more stake a node has, the more it is interested in the correct functioning of the system, since he's the one that would lose the most if the system stopped working or lost trust and the public decided not to use it.

PoS systems are based in a pseudorandom number generator, that will generate a deterministic value every time a block must be created. This value will be the same for all the nodes that have the same chain and have the same state. The system will provide a way to elect a node based on the generated value, and it will keep in mind the stake of every node. A

summary of this mechanism would be: every time a block has to be created, a lottery is made, where nodes have as many tickets as stake. Anyone could be elected, but nodes with higher stake have more chances to win the lottery.

The elected node, called the validator, will have to sign the created block so everyone can validate that the proposed block has been created by the winner of the lottery.

It is easier to enforce a block time, since we can directly control when a block is elected through the lottery. We can ensure no node is elected until a certain time, which will be calculated using the level of the node (the number of ancestors) and the initial date written in the genesis block.

5.2.2 Pros

- 1) PoS blockchains are susceptible to 51% attacks, since a node that owns 51% of the stake is going to be the major voice driving the consensus of the system. In this model, though, this actor would be the one who lost the most, so it wouldn't make sense for him to attack the network.
- 2) If the stake is a valuable asset, it will be harder by a single node to control a big part of it. Because the stake will be more distributed between than nodes than computational power would, the consensus is more decentralized that it would be in PoW.
- 3) No need to consume large quantities of electricity in order to secure a blockchain.
- 4) It has a more stable block time than PoW, since it is directly controlled through the lottery.

5.2.3 Cons

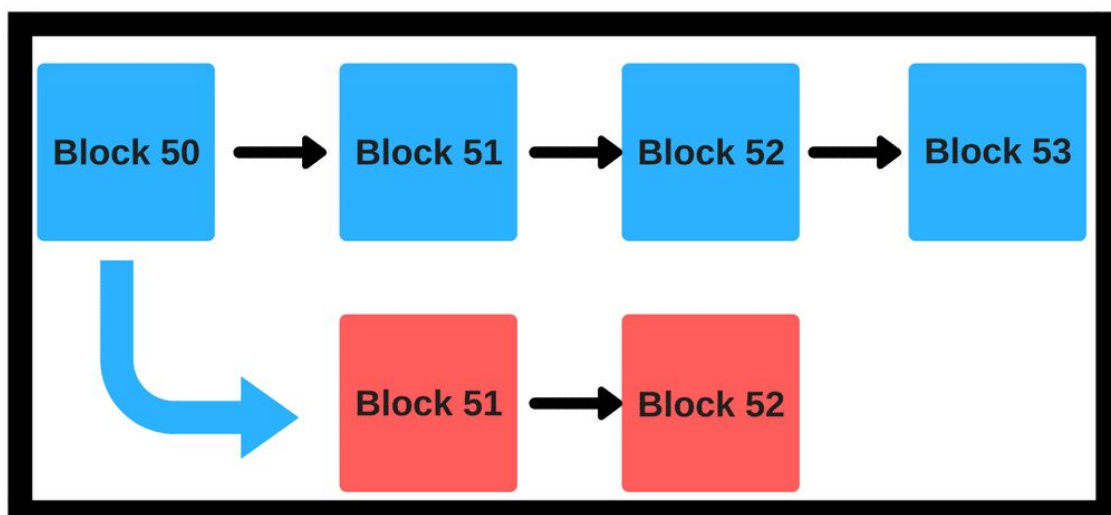


Figure 2: A forked blockchain

- 1) Because creating blocks has no real cost, there can be problems when there's a fork in the chain (Figure 2), since nothing deters nodes from trying to create blocks for every branch of the chain. This is called the "Nothing at Stake" problem. In comparison, nodes on PoW would decide to pick a single one of the branches, the one they believe to be correct, because the amount of resources they have is limited.

This is relevant because if nodes vote for each branch of the chain they know, they are actively preventing consensus from being achieved, and the chain could be hard-forked and never converge. This can be addressed with some mechanism that penalizes nodes that form blocks on multiple branches.

5.3 Federated Byzantine Agreement

The Stellar Consensus Protocol (SCP) is a consensus algorithm proposed by Dr. Mazières on 2015 [20] and belongs to a relatively new family of consensus algorithms called Federated Byzantine Agreement (FBA). In this section we will talk directly about SCP because at the moment is the only proven implementation of FBA that is truly decentralized.

While FBA was pioneered by the Ripple blockchain, it requires a network owner that manages a list of nodes that can create blocks. This means Ripple is a permissioned blockchain, so we won't be able to use its model on IPchain.

5.3.1 Explanation

In blockchain, there's a family of consensus algorithms that can achieve consensus through Byzantine Fault Tolerance (BFT) techniques. These techniques have been used on distributed systems for a long time, since they provide some resistance to ill-behaved nodes. Because blockchains are distributed systems, BFT techniques can be easily adopted as consensus algorithms.

The basic mechanism of BFT consensus algorithms is that all nodes dispose of a list of validator nodes. When consensus must be achieved, all validator nodes will propose blocks. Once blocks are proposed, all validator nodes will vote which is the block they want to add to the chain. If there is any tie, new votes will be started with the tied options. Eventually, one block will be elected and added to the chain. The problem with this kind of algorithms is that someone has to maintain the list of validators, which usually leads to some sort of centralized control. FBA algorithms actively try to apply BFT techniques while truly maintaining the list of validators in a decentralized way.

In SCP, every node N defines something called “quorum slice”, which is a subset of nodes of the entire system that also contains the node N itself. When a quorum slice (or simply slice) agrees on the validity of a block, the node N that defined the slice believes their result and goes along with it. A quorum slice is a subset that is enough to represent the consensus of the entire system for the node N . A node can have as many quorum slices as he wants, but only requires one slice to reach consensus. We can see a visual representation of a FBA in Figure 3, where each delimited area represents a slice.

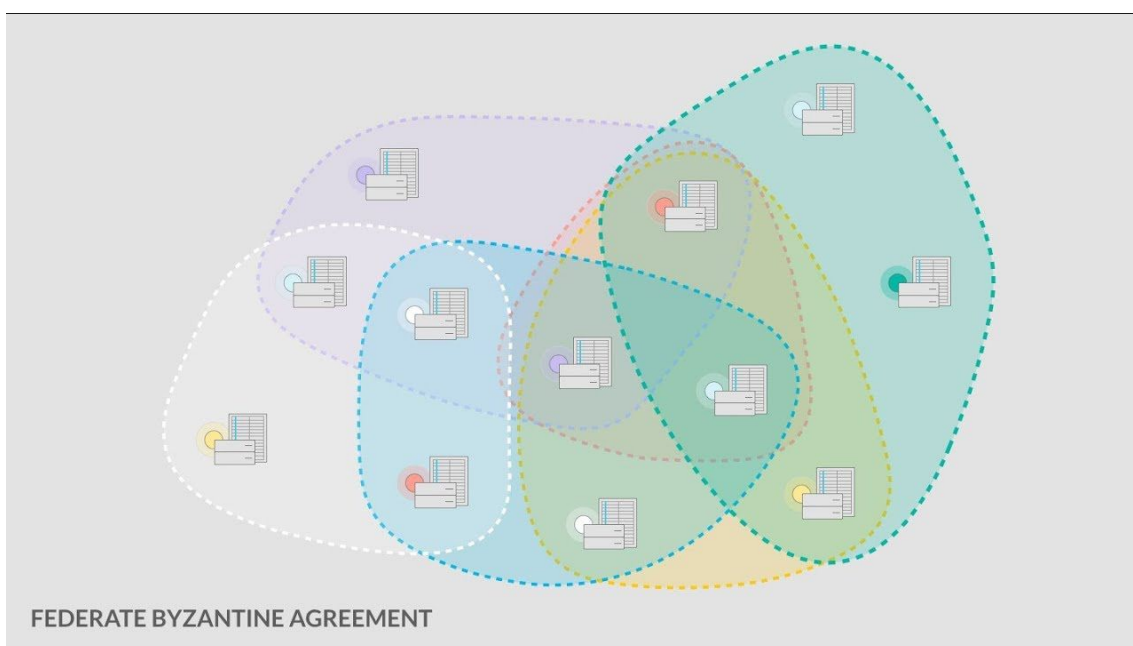


Figure 3: Quorum slices in FBA

Because of the configuration of SCP, it prefers security over liveness. This means a node won't be able to reach consensus until one of its slices agree on something. A consequence of this is that nodes on FBA won't see forks, they will always have one single chain.

This doesn't mean that forks can't happen in FBA. Because quorum slices are defined by users, some troublesome scenarios can appear. In the following figures, nodes will be represented by circles and arrows will represent the slices of every node.

- There is no intersection between the slices of two (or more) subgroups of the system. This would mean that chain from nodes v_1, v_2, v_3 could easily fork from the chain from nodes v_4, v_5, v_6 , as their consensus mechanisms would be completely isolated.

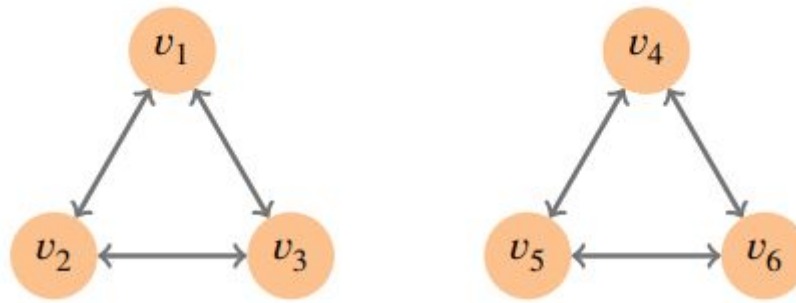


Figure 4: Slices with no intersection

- There is not enough nodes intersecting between the slices of two (or more) subgroups of the system. The intersecting nodes could easily behave maliciously and vote for different blocks in every subgroup, thus forcing a fork between the chain of each subgroup.

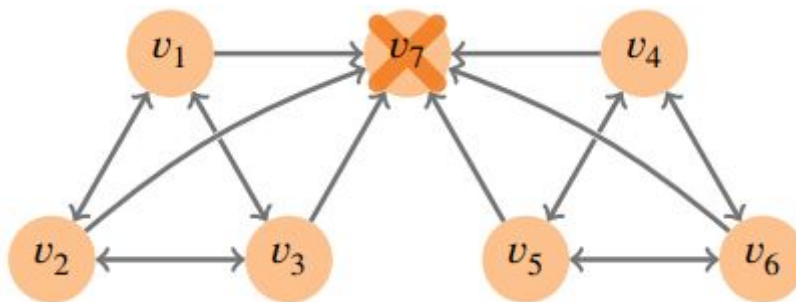


Figure 5: Slices with ill-behaved intersection

While nodes are free to pick their quorum slices, they have to be very cautious on the slices they pick. If slices aren't picked correctly, the security of the system could be compromised. There is also no easy way to know when the slices picked will provide safety to the system.

In a lot of cases where FBA is applied, the majority of nodes will pick very similar slices, since there will be some big actors on the system that everybody will trust (for example, if we were using FBA to implement an interbank blockchain, most spanish users would trust the same banks: La Caixa, Banco Santander, Banc Sabadell, etc). The consequence would be that, since everyone would trust the big actors, those would be the ones that would drive the consensus forward, while other nodes simply accept their judgement.

5.3.2 Pros

- 1) Consensus is reached through voting and deciding which votes a node is going to trust. This sounds like a more democratic way to make decisions in a decentralized system than PoW and PoS.
- 2) Because of previous point and the impossibility of forks on a single node, block time can be smaller than in other implementations. This means transactions in FBA will be processed a lot faster than in other consensus algorithms (3 to 5 seconds in SCP).

5.3.2 Cons

- 1) Creating slices is not easy. In theory nodes should form slices that represent their own trust relationships, but in practice a node will have to form slices that provide security to the system, so his trust relationships will be distorted for a greater good.
- 2) The entire system is based in graph theory, so it isn't as linear to code and test as other consensus algorithms.

6. Design

In this section we will discuss the design of the consensus algorithm developed for IPchain.

6.1 Discussion of researched algorithms

The first step to designing our consensus protocol is framing it in one of the families previously researched. We will first discuss each one of them separately, and we'll finally reach a conclusion on the family of our consensus protocol.

We first wanted to note that unlike cryptocurrencies, altering the state of IPchain will not provide a lot of benefits to a malicious party. Since IPchain will only act as a security layer for BGP, he could simultaneously attack both IPchain and a BGP router to reroute traffic from a certain ip prefix to himself. This way, he could analyze the incoming traffic to that prefix and negatively impact the true owner of the prefix, but he would have to spend a lot of resources to do so, and he wouldn't really obtain a lot in exchange, since the attack would be discovered and fixed in a few hours.

Also, ip prefixes are allocated in a block manner (as represented in Figure 6), where there's a central authority (IANA) that owns all the prefixes and distributes them in huge blocks to each RIR. RIRs allocate or delegate smaller blocks to ISP's, whom can finally assign them to users. IPchain will mimic this distribution too, and one node representing IANA will own all the prefixes on the genesis block, and distribute them as new blocks are inserted.

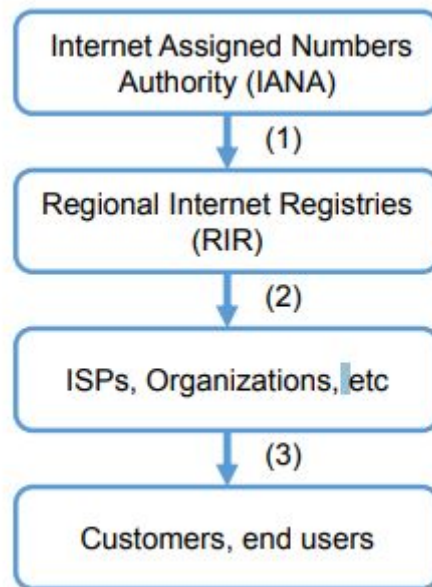


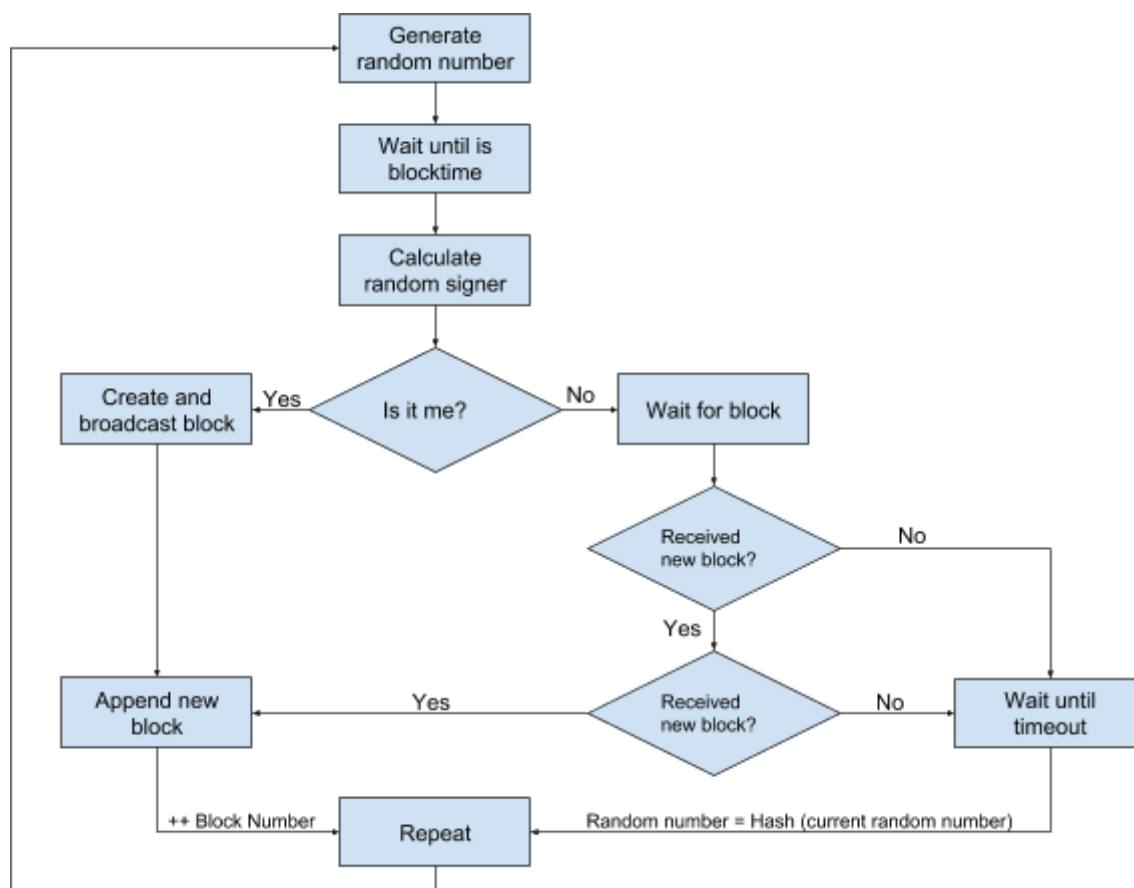
Figure 6: IP address allocation hierarchy

- 1) **Proof-of-work:** While PoW algorithms have been heavily tested, and we know for sure that they work, the fact that they require a huge amount of processing power and consume a lot of energy puts us off. Because attack IPchain doesn't provide a lot of benefit, all the security layer provided by the mining of blocks isn't really required. Because of these reasons, we would rather use any other consensus algorithm more fitting to IPchain requirements.
- 2) **Proof-of-stake:** PoS keeps all the good bits of PoW, while removing the mining part. As we have discussed previously, mining was an inconvenience to us, so we are completely fine with that. We also believe that the stake concept maps perfectly to the IP prefixes we will be handling in IPchain. While PoS hasn't been as heavily tested as PoW, there are some blockchain systems out there already using PoS consensus algorithms, so we can use those implementations as reference for our own. The only real problem with PoS is the "Nothing at Stake" problem, but we think it really isn't that troubling in IPchain. On the first place, only actors that own some IP prefixes will be able to create blocks, as they will be the only ones that have some stake. On the second place, because IPchain reflects real trust relationships (as seen in Figure 6), if an entity decided to act maliciously, he would be risking those relationships. It would also be easy for all the actors of the system to recognize who is acting maliciously and ignoring their block proposals.
- 3) **Federated Byzantine Agreement:** While FBA looked like a very promising concept, we believe that it's still too new and some concepts need to be refined. As we have

previously mentioned, we believe there's a problem with the way quorum slices have to be selected, so nodes would have to make a choice: either they create slices that reflect their trust relationships or they create slices that make the system secure. We also believe that this consensus algorithm is too complex when compared to PoS, and doesn't really provide a lot of benefits, since IPchain doesn't really require a high throughput.

After revising the researched consensus algorithms, we have decided to implement a protocol based in **Proof-of-Stake**.

6.2 The protocol



Since the protocol we have designed to implement is a classic Proof-of-Stake, we have decided to summarize the entire protocol in a scheme, so we can clearly see how it works. We will dive into details once we start the implementation, for now we only need to clear a couple points:

- We have added a max amount of time to wait for a block from the elected node. The timeout should give the validator enough time to recover from soft errors such as a bug in the code or small internet problems.

- It is possible that only some nodes receive a valid block, while others will skip this round and proceed to the next one. This means that forks could happen in this model, but given the fact that we are assuming all of the nodes in IPchain are well-intentioned, this is a really extreme situation. We have decided that at the moment, the only way to recover from a fork is restarting the node so he can catch up with the correct ones.
- If a round is skipped because a valid block was never received, we use a different mechanism to generate the following random number. This happens because the random number is generated from the current state of the chain, if we don't add a new block the random number generated will be the same as the previous round. If we assume that the previous elected node didn't create a block because he was unavailable, it doesn't make sense to elect him as a leader again. We will explain this in further detail on section 7.3.

7. Implementation

The most important part from our design is the generation of pseudorandom numbers (PRN). We require of a beacon that will generate a PRN that must be deterministic for a state of the chain, so all nodes that are in the same chain will obtain the same PRN when consulting their beacon.

Implementing this pseudorandom number generator (PRNG) is a complex problem that requires knowledge of cryptography and other specialized areas. Since the author of the project has no such knowledge, we have decided to use the same scheme as DFINITY, a blockchain implementation that also uses Proof-of-Stake. We chose DFINITY because we had both the whitepaper [21] and the source code [22], so it was possible to implement the PRNG by ourselves.

We will now proceed to explain some building blocks which we have implemented and used for our consensus algorithm.

7.1 Threshold signature

7.1.1 Explanation

A threshold signature is a cryptography primitive that works similar to public-key cryptography.

Following the example in Figure 7, Alice can use public-key cryptography to generate a matching public and private key (p and s). She can then generate a message M and sign it with her secret key s , producing a digital signature σ . She can now send the pair of M and σ to anyone else. Whomever receives the message can verify M and σ using the public key of Alice.

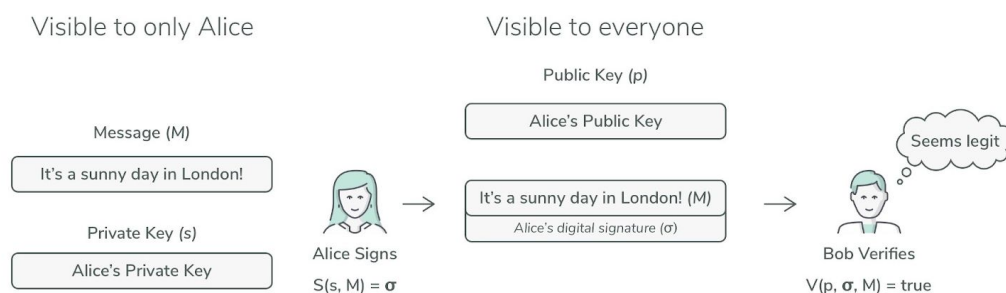


Figure 7: Example of public-key cryptography

While classic public-key cryptography is great, it is limited to a single sender. In certain cases, we want to send messages as a group. To avoid having a single member of the group tampering with the message, we can have several members sending the digital signature of the message and asking the receiver to only trust the message once he has received N signatures. We will call this value N threshold.

While the previously mentioned strategy would work, it is not very efficient and wouldn't work for decentralized scenarios. This is where threshold signatures can help us. A threshold signature is a special way of doing multiple signatures where given a group of participants and a threshold, each one of them is going to receive a share, and a public key will be generated for the group. In order to generate a valid signature to verify a message, the receiver needs to collect *threshold* (or more) different signature shares and combine them. A signature share of a message can be generated by a user of the group signing the message with their share. An example of threshold signature can be seen in Figure 8.

At least 5 signature shares can be combined into a single overall signature

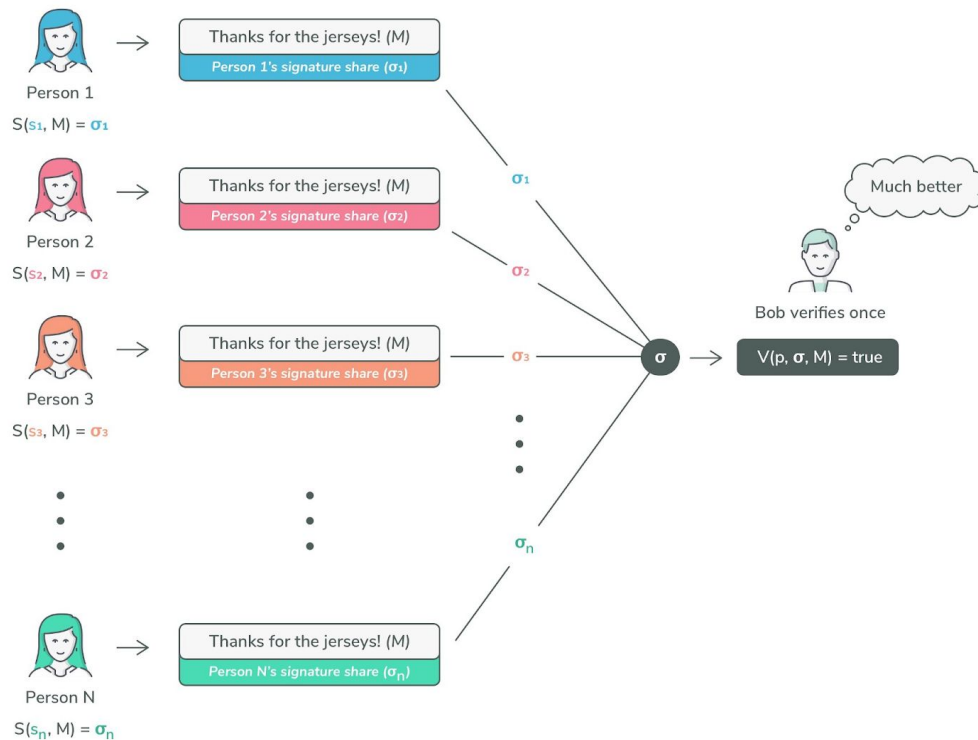


Figure 8: Example of threshold signature

7.1.2 Implementation

In order to implement our PNRG we used a threshold signature scheme called BLS. We picked this one because it is unique, deterministic, non-interactive and DKG-friendly.

- Deterministic: The signing function doesn't use randomness and will always give the same result.
- Unique: For every message and every public key there is only one signature that validates successfully.
- Non-interactive: The process of creating the group signature involves only a single round of one-way communication for each of the participants.
- DKG-friendly: It is possible to generate the keys required for the party without the help of a central authority. This will be explained in section 7.2.

The IPchain codebase is written in Python, but no native implementation for BLS was found. Because of this, and also to avoid unexpected compatibility issues, we decided to use the same library DFINITY uses [23].

We had to fork the library, create some bindings and wrap it in a python library, so the bindings can be replaced if a native implementation is ever built.

Our BLS API provides the following functions:

- *genKeys(id)*: Create a matching pair of public and private/secret key (used as seeds for other operations).
- *share(secKey, threshold, ids)*: Create shares for a defined group and threshold.
- *sign(msg, secKey)*: Create a digital signature for message *msg*.
- *verify(msg, sig, pubKey)*: Verify a message with the given signature and public key.
- *recover(ids, sigs)*: Generate the digital signature from the signature shares and the ids of those who generated them.
- *getPublicKey(sk)*: Get the matching public key of a secret key.
- *secretKeyShare(id, sKeys)*: Generate a share from an array of secret keys.
- *publicKeyShare(id, vVec)*: Generate a public key from a verification vector.
- *publicKeysEqual(pk1, pk2)*: Compares two public keys.
- *secretKeyAdd(sk1, sk2)*: Generates a secret key from two secret keys.
- *publicKeyAdd(pk1, pk2)*: Generates a public key from two secret keys.

A BLS sample program was created to test and validate how it works.

```
#!/usr/bin/env python2.7

import sys, os
sys.path.append(os.path.join(os.path.dirname(__file__), ".."))
import libs.bls_wrapper as bls

def main():

    myId = 1234
    m = "hello bls threshold signature"
    ids = [ 1, 5, 3]
    k = 2

    sk, pk = bls.genKeys()
```



```

if not sk:
    print("Error initializing bls")
    return

sm = bls.sign(m, sk)

if not bls.verify(m, sm, pk):
    print("Error verifying initial message")
    return

shares = bls.share(sk, k, ids)

sigs = []
for share in shares:
    aux = bls.sign(m, share["sk"])
    if not bls.verify(m, aux, share["pk"]):
        print("Error verifying message from id " +
str(share["id"]))
        return

    sigs.append(aux)

s = bls.recover(ids, sigs)
if not bls.verify(m, s, pk):
    print("Recover failed")
    return

print("Recover is OK")

if __name__ == "__main__":
    main()

```

7.2 Distributed Key Generation

7.2.1 Explanation

Distributed key generation (DKG) is a process through which a threshold signature scheme can be completed for a group without the intervention of a centralized authority. Once the DKG is completed, every member will have a share and a public key for the group will be generated.

7.2.2 Implementation

DFINITY source code contains a library written in Javascript that provides primitives required to perform a DKG. Because the library was short, we ported it to Python.

The DKG API provides the following functions:

- *generateContribution(threshold, ids)*: it will generate a contribution for every id and a verification vector used to verify those contributions.
- *verifyContributionShare(id, contribution, vVec)*: it will verify that the contribution sent by the node *id* is correct, using a verification vector.
- *generateShare(contributions)*: it will combine several contributions to generate a share for the local node.
- *addVerificationVectors(vVecs)*: it will combine several verification vectors to generate the public key of the group.

A DKG sample program was created to test and validate how it works. This program emulates a distributed network but is actually executed in a single machine to simplify the testing.

```
#!/usr/bin/env python2.7

import sys, os
sys.path.append(os.path.join(os.path.dirname(__file__), ".."))
import libs.bls_wrapper as bls
import dkg as dkg
from random import randint
import argparse

def dkgSetup(ids, threshold):
    members = []
    for member in ids:
        secKey, _ = bls.genKeys(member)
        members.append({
            "originalId": member,
            "id": secKey,
            "receivedShares": [],
            "secretKeyShare": None
        })

    print("Beginning the secret instantiation round...")
    print("\tUsing members: " + " ".join(map(str, ids)))
    print("\tUsing threshold of: " + str(threshold))
```

```

vVecs = []
for _ in members:
    verificationVector, secretKeyContribution =
dkg.generateContribution(threshold, [ m["id"] for m in members ] )
    vVecs.append(verificationVector)

    for i, contrib in enumerate(secretKeyContribution):
        member = members[i]
        b = dkg.verifyContributionShare(member["id"], contrib,
verificationVector)
        if not b:
            print("Invalid share!")
            return

        member["receivedShares"].append(contrib)

for member in members:
    sk = dkg.addContributionShares(member["receivedShares"])
    member["secretKeyShare"] = sk

print("-> Secret shares have been generated")

groupsvVec = dkg.addVerificationVectors(vVecs)
print("-> Verification vector computed")
print("Resulting group public key is " + (groupsvVec[0]) + "\n")

return members, groupsvVec

def dkgTest(msg, members, vVec, threshold):
    groupsPk = vVec[0]
    sigs = []
    signerIds = []

    print("Testing signature on message \"" + msg + "\"")

    while len(sigs) < threshold:
        i = randint(0, len(members)-1)
        if members[i]["id"] in signerIds:
            continue

        skShare = members[i]["secretKeyShare"]
        print("-> Member " + str(members[i]["originalId"]) + " signs
with share: " + skShare)
        sig = bls.sign(msg, skShare)
        sigs.append(sig)

```

```

        signerIds.append(members[i]["id"])

    groupsSig = bls.recover(signerIds, sigs)
    print("Resulting sig: " + groupsSig)

    verified = bls.verify(msg, groupsSig, groupsPk)
    print(("\\033[92m" if verified else "\\033[91mNOT ") + "VERIFIED
\\033[0m\\n")

    return groupsSig

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-m", help="Ids of members of the dkg",
nargs="*", default=[ 10314, 30911, 25411, 8608, 31524, 15441,
23399], type=int)
    parser.add_argument("-th", help="Threshold of the threshold
signature that will be setup", nargs="?", type=int)
    parser.add_argument("-rr", help="Amount of times a dkg round
will be repeated (for the same message)", nargs="?", default=2,
type=int)
    parser.add_argument("-nr", help="Amount of rounds, where the
message from the previous round will be signed", nargs="?",
default=5, type=int)
    args = parser.parse_args()

    ids = args.m
    threshold = args.th if args.th else len(ids)/2 + len(ids)%2
    numRounds = args.nr
    roundRepeat = args.rr

    members, vVec = dkgSetup(ids, threshold)

    msg = "This is a dkg sample"
    for i in range(numRounds):
        print("ROUND " + str(i+1))
        print("-----")
        roundMsg = msg
        for j in range(roundRepeat):
            msg = dkgTest(roundMsg, members, vVec, threshold)

if __name__ == "__main__":
    main()

```

This program will setup a DKG between members 10314, 30911, 25411, 8608, 31524, 15441 and 23399, then all members will sign an initial hardcoded message with their shares, we will collect all the signature shares and validate them using the groups public key.

We will execute the sample twice (Figures 9 and 10), repeating the same process twice for every DKG setup, so we can point out a couple interesting points on the DKG behaviour.

- 1) Every time a DKG is completed, the group public key and the shares of every node are different (they are random values). The resulting digital signature of the original message is different too.
- 2) No matter which signature shares we recover, the resulting digital signature is always the same.

```

jan@janlaptop ~/Projects/tfg/blockchain/Consensus (dht) » ./examples/dkg-sample.py -rr 2 -nr 1
Beginning the secret instantiation round...
    Using members: 10314 30911 25411 8608 31524 15441 23399
    Using threshold of: 4
-> Secret shares have been generated
-> Verification vector computed
Resulting group public key is 1 0x1ae0157fb3bd0e4d65c4179098d76ee5b5e24059981103ae7b57e99f1804477d 0x13fa39c1baaf2c0961
6b12273255436fa9b15f3da363b230475b1ee1e2f69528 0x515f0259f56691c1f1dee4e1ba3663adc24a1d712146ea1b0d91a6454d725d9 0x17a9
ea74a2bb0e04f38e955190337e79c4aa6153e6f2fd6a8a5ab3e9c86b1087

ROUND 1
-----
Testing signature on message "This is a dkg sample"
-> Member 8608 signs with share: 0x1dd50e830be55e8601dbeb87aed20dc840911b96796d6effb7a0c769ed5c2cca
-> Member 15441 signs with share: 0xc3c7a73c1201653e94da5d51a03553833fd84cc62555440a9dac6b1a1b2eb15
-> Member 30911 signs with share: 0xc3c7a73c1201653e94da5d51a03553833fd84cc62555440a9dac6b1a1b2eb15
-> Member 25411 signs with share: 0x3b045b760bab3f792a0af486df3dd53c52b58175068110afda0f90ca8cf7be
Resulting sig: 1 0x823c4766ab9df4f0f60bc370bfe5f5a242e8160c77dbb4682d8c443f4cc2edf 0x593065f377bdb5ebf12e7a23bab7d2b9bb
57e986733dcc8afff331143736908
VERIFIED

Testing signature on message "This is a dkg sample"
-> Member 30911 signs with share: 0xc3c7a73c1201653e94da5d51a03553833fd84cc62555440a9dac6b1a1b2eb15
-> Member 8608 signs with share: 0x1dd50e830be55e8601dbeb87aed20dc840911b96796d6effb7a0c769ed5c2cca
-> Member 23399 signs with share: 0x297f7e6d5c6fab9045c56155b337c9aaf062c3abb9093ed195be0942950eeefb
-> Member 10314 signs with share: 0x1342ff8df4ede776eb6f307f392bab1e4593c97d1107675499a5660f221ce6d6
Resulting sig: 1 0x823c4766ab9df4f0f60bc370bfe5f5a242e8160c77dbb4682d8c443f4cc2edf 0x593065f377bdb5ebf12e7a23bab7d2b9bb
57e986733dcc8afff331143736908
VERIFIED

```

```

jan@janlaptop ~/Projects/tfg/blockchain/Consensus (dht) » ./examples/dkg-sample.py -rr 2 -nr 1
Beginning the secret instantiation round...
    Using members: 10314 30911 25411 8608 31524 15441 23399
    Using threshold of: 4
-> Secret shares have been generated
-> Verification vector computed
Resulting group public key is 1 0x16033b9121297f7f61b193971fe06bdeba731781a228e85bbd73b080e3a724 0x13bad584a656e17697
d479ad55909f717064b5a7926fafb3937cedf2f78a7f0a 0x2fa91c32635f3d6e22a62bc86eb91d99efdddcf3c8a6210eedf5280765611d 0x1255
322170bcd629be4bf97b6b0dbdd96767afdbf08a16a80089ff9cb26285a19

ROUND 1
-----
Testing signature on message "This is a dkg sample"
-> Member 31524 signs with share: 0xce05911378bc2f5ee8a5bd3ba4d8248f588bf7fc53e08563fbd3ac92e03d619
-> Member 8608 signs with share: 0x10b62924db48b916805dfb7cbaeea27c8d7a4aa5b08b1278bdb68e757b0c8e66
-> Member 25411 signs with share: 0x39cbc13d0a7513c6949068ef8cf3b1def8e72b361f70ef6b053df0a18578b03
-> Member 30911 signs with share: 0xd7402979d23a9114ee6c4c00c8e22838fcd717dba9a5fbb6966edca2c1dd7e8
Resulting sig: 1 0x1b2a4017a92300f92a4eb883a3e1e0dcec60365bfbe7fa96c05f7ba68ae548b8 0x23bdfd5878b597ac3b52a8af448505d4f
cb15f2facdd6179ca918501d8eadd6d
VERIFIED

Testing signature on message "This is a dkg sample"
-> Member 15441 signs with share: 0x12654235dbddab924c1f55dc75c953b2a45c2a2951bc69d97aed599251c89865
-> Member 31524 signs with share: 0xce05911378bc2f5ee8a5bd3ba4d8248f588bf7fc53e08563fbd3ac92e03d619
-> Member 25411 signs with share: 0x39cbc13d0a7513c6949068ef8cf3b1def8e72b361f70ef6b053df0a18578b03
-> Member 30911 signs with share: 0xd7402979d23a9114ee6c4c00c8e22838fcd717dba9a5fbb6966edca2c1dd7e8
Resulting sig: 1 0x1b2a4017a92300f92a4eb883a3e1e0dcec60365bfbe7fa96c05f7ba68ae548b8 0x23bdfd5878b597ac3b52a8af448505d4f
cb15f2facdd6179ca918501d8eadd6d
VERIFIED

```

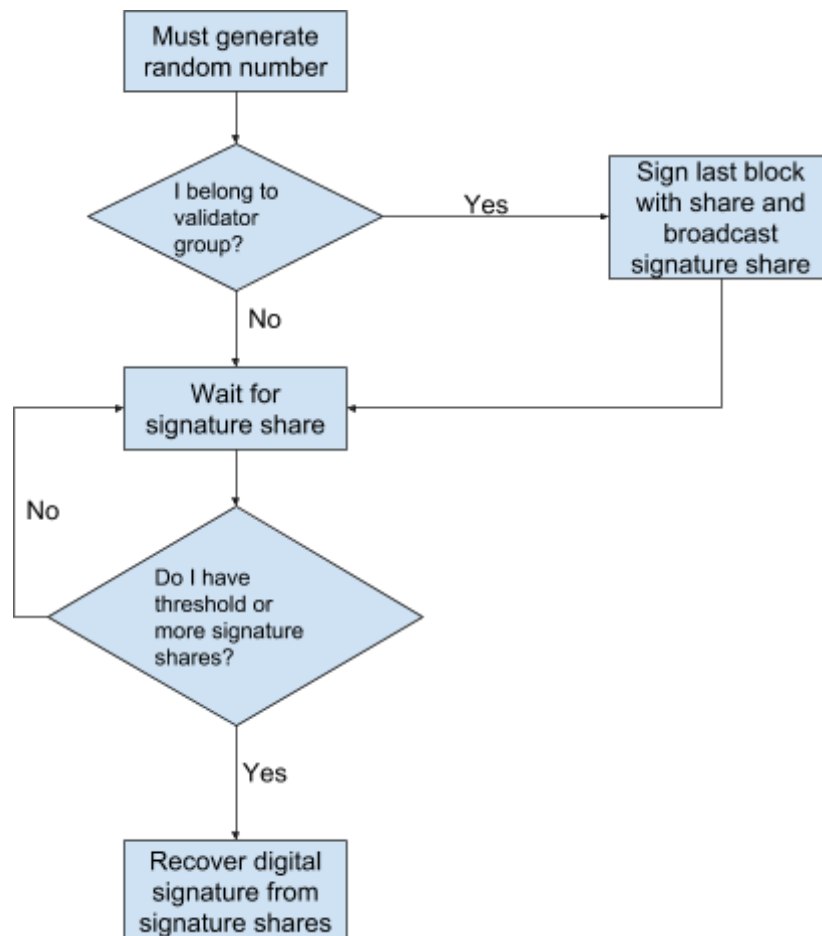
Figures 9 and 10: Executions of DKG sample

7.3 Pseudorandom number generation

Now that BLS and DKG have been explained and implemented, we can explain how the PRNG works. In order to generate PRN we first need to run a DKG process to setup our blockchain system.

Once the DKG is completed, the system is ready to generate random numbers.

To generate a random number we will use the signature of the last block in the chain. The process is described in the following scheme:



The final signature share will be verified using the groups public key. If this verification fails after all the signature shares have been received it means that there are not enough nodes that have the same chain state as the node that is trying to generate a random number. At this point, all the node can do is restart to try to recover. Some improvements to this process will be discussed in section 10.

To setup a DKG, we need to define a list of members that will participate in the process. Given the facts that some members could be disconnected at the time of DKG and that the blockchain is public and new members could appear at any time, the DKG cannot be definitive. In order to eventually include these members, a new DKG has to be done every certain time. This is a configuration parameter that will be discussed in section 8.2.

The time the DKG will require will scale linearly with the amount of participating members. Because of this, the amount of members that will be used is a configuration parameter that will be discussed in section 8.2.

Completing a DKG will provide the participating members with the group public key, but we need a way for members that haven't participated in the DKG to know this public key. We will use the blockchain to do this: when a DKG is completed, a special block containing the groups public key will be inserted into the blockchain. The responsible of creating this block will be one node randomly selected from all the participating members of the DKG.

As an added feature, every node that participated in the DKG will store their share locally with some sort of encryption. This way, if the node needs to restart, it can recover its state.

8. Results

We have had an unexpected problem with the planning of this project. We never considered the generation of pseudorandom numbers in a distributed environment, since at the time of the planning we did not know how Proof-of-Stake consensus algorithms worked, but it proved to be a topic as complex as the implementation of a consensus algorithm. Since these are state-of-the-art techniques that extensively use advanced cryptography, and the author is not an expert in these fields, the adaptation of DKG and BLS took more time than expected. This meant we didn't have enough time to implement all this model in the IPchain codebase.

Instead, we have decided to build prototypes that will allow us to test our design. Another developer will have to integrate our prototype into IPchain.

Our prototype has been built following the DKG sample appended in section 7.2. In order to emulate a P2P network, we have built a broker using a networking library called ZeroMQ. Messages are sent through TCP. We also implemented a CLI that allows us to decide when to setup a DKG or generate a new random number.

This is the prototype code that represents one node:

```
#!/usr/bin/env python2.7

import sys, os
sys.path.append(os.path.join(os.path.dirname(__file__), "../.."))
import libs.bls_wrapper as bls
import dkg as dkg
import zmq
import signal
import json
import argparse
import logger

signal.signal(signal.SIGINT, signal.SIG_DFL);

log = None
sk = None
groupPk = None
state = "ipchain"
sigs = None
sigIds = None
```



```

addrs = {}
brokerAddr = None

def init(oid, threshold, pport, sport):
    global log, sk, addrs, brokerAddr
    log = logger.setup_custom_logger(str(oid))
    members = {}
    oids = []

    for line in open("examples/tcp-utils/members.txt").readlines():
        lines = line.split(' ')
        id = int(lines[0])
        addr = lines[1].rstrip("\n")
        oids.append(id)
        if not brokerAddr:
            brokerAddr = addr
        addrs[id] = addr

    ctx = zmq.Context()
    sub = ctx.socket(zmq.SUB)
    sub.connect("tcp://%s:%s" % (brokerAddr, sport))
    sub.setsockopt(zmq.SUBSCRIBE, "")

    pub = ctx.socket(zmq.PUB)
    pub.connect("tcp://%s:%s" % (brokerAddr, pport))

    socket = ctx.socket(zmq.REP)
    socket.bind("tcp://*:%s" % oid)

    poller = zmq.Poller()
    poller.register(socket, zmq.POLLIN)
    poller.register(sub, zmq.POLLIN)

    end = False

    log.info("%d: Started communications..." % oid)
    while not end:
        socks = dict(poller.poll())
        if sub in socks and socks[sub] == zmq.POLLIN:
            msg = sub.recv()
            type = msg.split("_", 1)[0]
            if type == "setup":
                setup(members, oid, oids, threshold)
            elif type == "randomNum" and sk is not None:

```

```

        pub.send(genNewSig(oid))
    elif type == "sig":
        handleSig(json.loads(msg.split("_", 1)[1]),
threshold, members)

    if socket in socks and socks[socket] == zmq.POLLIN:
        msg = json.loads(socket.recv())
        topic = msg["topic"]
        if topic == "contrib":
            receiveContribution(msg, members, oid)
            socket.send("OK")

def genNewSig(m_oid):
    global state, sk, sigs, sigIds
    sig = bls.sign(state, sk)
    sigs = []
    sigIds = []

    return ("sig_" + json.dumps({
        "oid": m_oid,
        "sig": sig
    })))

def handleSig(msg, threshold, members):
    global sigs, groupPk, state, sigIds

    if not groupPk:
        return

    sigs.append(msg["sig"]);
    sigIds.append(members[msg["oid"]]["id"]);

    log.info("Received secretShare %s" % msg["sig"])

    if (len(sigs) >= threshold):
        groupsSig = bls.recover(sigIds, sigs)

        if bls.verify(state, groupsSig, groupPk):
            state = groupsSig
            log.info("Verified sig %. Updating state..." %
groupsSig)

def receiveContribution(msg, members, m_oid):
    oid = msg["oid"]
    contrib = msg["contrib"]

```

```

vVec = msg["vvec"]

if dkg.verifyContributionShare(members[m_oid]["id"], contrib,
vVec):
    members[oid]["receivedShare"] = contrib
    members[oid]["vvec"] = vVec
    log.info("Received valid share from member %s" % oid)
else:
    log.info("Received invalid share from member %s" % oid)

if allSharesReceived(members):
    global sk, groupPk
    sk = dkg.addContributionShares( [ member["receivedShare"]
for _,member in members.iteritems() ])
    groupsvVec = dkg.addVerificationVectors( [ member["vvec"]
for _,member in members.iteritems() ])
    groupPk = groupsvVec[0]
    log.info("DKG setup completed")
    log.info("Resulting group public key is " + groupPk + "\n")

def setup(members, m_oid, oids, threshold):
    members.clear()
    for oid in oids:
        secKey, _ = bls.genKeys(oid)
        members[oid] = {
            "id": secKey,
            "receivedShare": None,
            "vvec": None
        }

    vVec, skContrib = dkg.generateContribution(threshold,
                                                [ member["id"] for
_,member in members.iteritems() ] )

    i = 0
    for oid, member in members.iteritems():
        if oid == m_oid:
            members[m_oid]["vvec"] = vVec
            members[m_oid]["receivedShare"] = skContrib[i]
        else:
            sendMsg(oid, {
                "oid": m_oid,
                "vvec": vVec,
                "topic": "contrib",
                "contrib": skContrib[i]

```

```

        })
        i += 1

def allSharesReceived(members):
    for _, member in members.iteritems():
        if not member["receivedShare"]:
            return False

    return True

def sendMsg(to, data):
    global addr
    context = zmq.Context()
    socket = context.socket(zmq.REQ)
    socket.connect("tcp://%s:%d" % (addr[to], to))
    socket.send(json.dumps(data), zmq.NOBLOCK);

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-id", help="Id of this node",
        required=True, type=int)
    parser.add_argument("-s", help="Subscribe broker socket",
        required=True, type=int)
    parser.add_argument("-p", help="Publish broker socket",
        required=True, type=int)
    parser.add_argument("-t", help="Threshold", required=True,
        type=int)
    args = parser.parse_args()
    init(args.id, args.t, args.p, args.s)

```

8.1 Performance

Ideally we would dispose of hundreds of computers to test our prototype. Unluckily, we only have five computers with a two core processor each one. To run our tests we will host several nodes in a single machine, so we will have to adapt the way we gather and interpret statistics.

While the statistics we gather can't be taken very seriously, they will give us an idea of the real results we would get in a distributed system formed of a lot of computers.

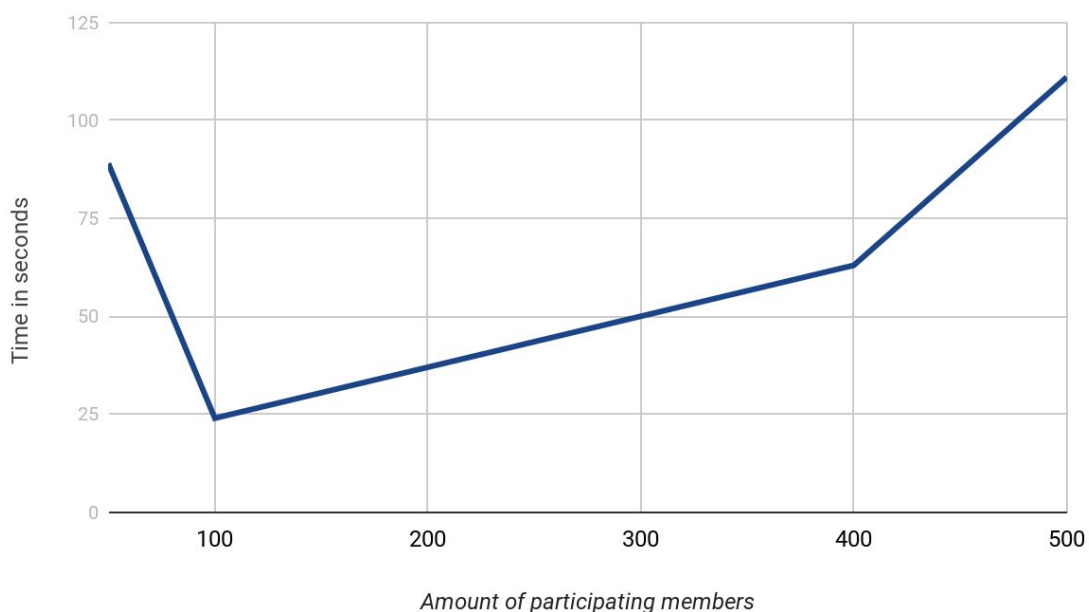
8.1.1 Time to setup DKG

Analyzing the prototype code, we can see that when a node does a DKG setup with N members and a threshold k , it generates N contributions and sends one contribution to every other member of the group (so it sends N messages). Once he has received N contributions and verification vectors, it combines them all to generate one share and the groups public key.

We have observed that the group public key computation is a cpu-intensive operation, and if we try to perform it once for every member hosted in the machine, the setup DKG time increases heavily. This happens because there are only two cores on the machine, so the group public key computations of all the nodes are performed sequentially (there are only two parallel executions). In order to fix this, we will still send N contributions per node hosted on a machine, but we will modify the code so that only one of those nodes will perform the posterior computations.

Also, because some nodes will be in the same machine, the latency from sending a tcp package to the localhost will be less than 1 ms. If the nodes were in different machines, the latency would be between 30 and 100 ms. This difference in latency wouldn't really affect the time to setup DKG greatly because it will only affect the time that it takes to distribute the contributions, so we won't take it into account. We expect this difference to be compensated by the fact that only two cores are used to process the amount of messages of a lot of nodes.

Time to complete DKG setup



8.1.2 Time to generate random number

8.2 Parameter recommendations

In this section we will discuss some configuration parameters that will alter the functioning of the blockchain.

8.2.1 Duration of the DKG setup

Because the participating members must be defined when doing a DKG, nodes connected after the DKG is completed or nodes that were disconnected during the setup won't be able to participate in the PRNG process. We believe that the DKG should be performed again after a certain number of blocks to include these new nodes. This amount of blocks will depend on the block time of the system. At the moment, the block time is 5 minutes, so we believe the DKG should be performed again after 50 blocks.

We picked 50 because this is roughly once every 6 hours, which is often enough for nodes to adhere to the PRNG process, but not too often to interfere with the consensus.

8.2.2 Amount of participant members on DKG

We should pick a value for this parameter that gives us a DKG setup time lower than the block time of the system. We should also keep in mind that the higher the amount of participants, the higher the chance to have some problem when doing the DKG. We can use the results of section 8.1.1 for this decision.

We believe that 100 is a good number, since the doing a DKG setup should be safe enough, but it still represents a representative chunk of the IPchain expected population.

8.2.3 Threshold

This value must be expressed as a percentage, and it represents the amount of shares from the same state a node must receive in order to generate a pseudorandom number. The higher this value is, the most improbable forks will be, but setting it too high could impact the ability to reach consensus (for example, at 100%, every time one node has a network problem or disconnects, consensus will be impossible).

If a node doesn't reach this threshold, he won't be able to generate a PRN and will be stuck until restart. Good values could be 51% and 66%.

9. Conclusions

The objectives of this project were:

- Study the currently known, tested and proved consensus algorithms that are used in blockchain solutions, identify their trade-offs and determine which one fulfills the requirements of IPchain the most.
- Study the existing IPchain codebase and implement the designed consensus algorithm on it.
- Test the implemented consensus algorithm, verifying it works correctly for an emulation of the BGP protocol. Also verify that non-functional requirements are fulfilled.

We believe this report contains a summary of the most relevant consensus algorithms at this time. We have also described their trade-offs and decided that Proof-of-Stake was the one that accommodated the needs of IPchain the most.

A consensus algorithm that fits the needs of IPchain has been successfully designed, and a prototype has been built. Unfortunately, it hasn't been possible to implement the consensus algorithm in the existing codebase of IPchain, but the hardest part is already completed. Another developer will have to integrate the results of this project.

It hasn't been possible to test our consensus algorithm in an emulation of the BGP protocol, because we haven't been able to implement the blockchain itself. We have used a prototype to verify that our design fulfills the non-functional requirements of IPchain (good enough performance, can adjust to desired block-time, ...).

Overall, we think we have done a pretty good job with this project, although it could have gone slightly better.

10. Future improvements

In this section we will describe some things that could be improved on the design described in this report. They were not implemented because they took too much time and the problem they solved wasn't important enough.

- Right now, if a node forks from others, he won't be able to recover its state to match the main chain. In order to fix this, the node must be restarted. It would be nice to provide an alternative to this.
- Because blockchain nodes use P2P to communicate, all the nodes can see every message. There are certain messages that have a single receiver, so they should be encrypted using public-key cryptography. We know that contributions should be encrypted, but there might be other messages that need encryption too.
- A public key for the contribution generated by a node can be obtained through the verification vector of that node. If we knew how to obtain this public key, we would be able to validate the signature shares received, so the entire PRNG process would be a bit more robust. We think DFINITY does this too, so the answer should be in their source code.
- To further prevent the "Nothing-at-Stake" problem, it could be a nice idea to record somewhere if a node is detected to be validating blocks for several branches of the chain at the same time.
- An improvement to the DKG setup would be taking the chain state into account. This way, if a node completes the DKG setup correctly, it knows for sure there are enough nodes with the same state, so he should be able to generate a valid signature share. Also, the amount of signature shares from nodes with a different chain will be reduced. This would make the PRNG process slightly more robust.
- Because of a limitation in the BLS library we use, we can only use integer node identifiers to setup the DKG. This is troublesome because the identifiers on the actual codebase are 40 byte long hex strings. Some information needs to be lost, because integers are 32 bytes. There's a chance two different hex strings are casted to the same integer, which will give problems with the threshold signature scheme. To fix this, we need to dive deep into the BLS library and understand how it works.

References

- [1] Ryan Singel, Pakistan's accidental youtube re-routing exposes trust flaw in net, February 2008. <https://www.wired.com/2008/02/pakistans-accid/>
- [2] Andy Greenberg, Hacker redirects traffic from 19 internet providers to steal bitcoins, July 2014. <https://www.wired.com/2014/08/isp-bitcoin-theft/>
- [3] Dyn Guest Blogs, Internet-wide catastrophe - last year, December 2005. <https://dyn.com/blog/internetwide-nearcatastrophela/>
- [4] J. Paillise, M. Ferriol, E. Garcia, H. Latif, C. Piris, A. Lopez, B. Kuerbis, A. Rodriguez-Natal, V. Ermagan, F. Maino, A. Cabellos, IPchain: Securing IP Prefix Allocation and Delegation with Blockchain, May 2018. <https://arxiv.org/pdf/1805.04439.pdf>
- [5] R. Bush, Internet Initiative Japan, R. Austein, Dragon Research Labs, The Resource Public Key Infrastructure (RPKI) to Router Protocol, January 2013. <https://tools.ietf.org/html/rfc6810>
- [6] Regional Internet Registries Statistics https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/World/ASN-ByNb.html
- [7] Ripe Network Coordination Center, Statistics. <http://certification-stats.ripe.net/>
- [8] M. Wählisch, R. Schmidt, T. Schmidt, O. Maennel, S. Uhlig, G. Tyson, RPKI: The Tragic Story of RPKI Deployment in the Web Ecosystem, November 2015. <https://dl.acm.org/citation.cfm?id=2834050.2834102>
- [9] S. Goldberg, Why is it taking so long to secure internet routing? August 2014. <https://dl.acm.org/citation.cfm?doid=2668152.2668966>
- [10] A. Hari, T. V. Lakshman, The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet, 2016. <http://web.kaust.edu.sa/Faculty/MarcoCanini/classes/CS390G/S17/papers/InternetBlockchain.pdf>
- [11] Q. Xing, B. Wang, X. Wang, POSTER: BGPCoin: A Trustworthy Blockchain-based Resource Management Solution for BGP Security, November 2017. <https://dl.acm.org/citation.cfm?id=3138828>
- [12] S. Angieri, A. García-Martínez, B. Liu, Z. Yan, C. Wang, M. Bagnulo, An experiment in distributed Internet address management using blockchain, July 2018. <https://arxiv.org/abs/1807.10528>
- [13] A. de la Rocha Gómez-Arevalillo, P. Papadimitratos, Blockchain-based Public Key Infrastructure for Inter-Domain Secure Routing, May 2017. <https://people.kth.se/~papadim/publications/fulltext/SBTM-blockchain-secure-routing.pdf>
- [14] BGPMon, Route Monitoring Services. <https://bgpmon.net/services/route-monitoring/>

- [15] ThousandEyes, BGP and Route Monitoring Services.
<https://www.thousandeyes.com/solutions/bgp-and-route-monitoring>
- [16] IRR, The Internet Routing Registry. <http://www.irr.net/>
- [17] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, October 2008.
<https://bitcoin.org/bitcoin.pdf>
- [18] Alex de Vries, Bitcoin's Growing Energy Problem, May 2018.
[https://www.cell.com/joule/fulltext/S2542-4351\(18\)30177-6](https://www.cell.com/joule/fulltext/S2542-4351(18)30177-6)
- [19] QuantumMechanic, Proof of stake instead of proof of work, July 2011.
<https://bitcointalk.org/index.php?topic=27787.0>
- [20] David Mazières, The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus,, April 2015. <https://www.stellar.org/papers/stellar-consensus-protocol.pdf>
- [21] Timo Hanke, Mahnush Movahedi and Dominic Williams, DFINITY Technology Overview Series: Consensus System. <https://arxiv.org/pdf/1805.04548.pdf>
- [22] DFINITY, DFINITY source code. <https://github.com/dfinity>
- [23] MITSUNARI Shigeo, BLS source code. <https://github.com/herumi/bls>
-
- [Figure 1] <https://blockgeeks.com/guides/what-is-hashing/>
- [Figure 2] <https://blockgeeks.com/guides/blockchain-consensus/>
- [Figure 3] <https://www.youtube.com/watch?v=X3Gj2nOZCNM>
- [Figures 4 & 5] <https://www.stellar.org/papers/stellar-consensus-protocol.pdf>
- [Figure 6] <https://arxiv.org/pdf/1805.04439.pdf>
- [Figure 7 & 8] <https://blog.keep.network/threshold-signatures-ff2c2b98d9c7>